Hexagonal Architecture - Package Structure

Version: 1.0
Date: 2025-11-26
Status: Implementation Ready

---

1. Maven Project Structure

```
signature-router/
├── pom.xml
├── README.md
├── .mvn/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── bank/
│   │   │           └── signature/
│   │   │               ├── SignatureRouterApplication.java
│   │   │               │
│   │   │               ├── domain/                    # DOMAIN LAYER (Pure
Business Logic)
│   │   │               │   ├── model/
│   │   │               │   │   ├── aggregate/
│   │   │               │   │   │   └── SignatureRequest.java
│   │   │               │   │   ├── entity/
│   │   │               │   │   │   ├── SignatureChallenge.java
│   │   │               │   │   │   ├── RoutingRule.java
│   │   │               │   │   │   └── ConnectorConfig.java
│   │   │               │   │   ├── valueobject/
│   │   │               │   │   │   ├── TransactionContext.java
│   │   │               │   │   │   ├── Money.java
│   │   │               │   │   │   ├── SignatureDecision.java
│   │   │               │   │   │   ├── ProviderResult.java
│   │   │               │   │   │   ├── RoutingEvent.java
│   │   │               │   │   │   └── CustomerId.java
│   │   │               │   │   └── enums/
│   │   │               │   │       ├── SignatureStatus.java
│   │   │               │   │       ├── ChallengeStatus.java
│   │   │               │   │       ├── ChannelType.java
│   │   │               │   │       └── ProviderType.java
│   │   │               │   │
│   │   │               │   ├── service/                # Domain Services
│   │   │               │   │   ├── RoutingService.java
│   │   │               │   │   ├── RoutingServiceImpl.java
│   │   │               │   │   ├── ChallengeService.java
│   │   │               │   │   ├── ChallengeServiceImpl.java
│   │   │               │   │   └── FallbackStrategyService.java
│   │   │               │   │
│   │   │               │   ├── port/                    # Hexagonal Ports
(Interfaces)
│   │   │               │   │   ├── inbound/            # Driving Ports (Use
Cases)
│   │   │               │   │   │   ├── StartSignatureUseCase.java
│   │   │               │   │   │   ├── ConfigureRuleUseCase.java
│   │   │               │   │   │   ├── RetryWithFallbackUseCase.java
│   │   │               │   │   │   ├── QuerySignatureUseCase.java
│   │   │               │   │   │   └── ManageConnectorUseCase.java
│   │   │               │   │   │
│   │   │               │   │   └── outbound/           # Driven Ports
(Dependencies)
│   │   │               │   │       ├── SignatureRequestRepository.java
│   │   │               │   │       ├── RoutingRuleRepository.java
│   │   │               │   │       ├── ConnectorConfigRepository.java
│   │   │               │   │       ├── AuditLogRepository.java
│   │   │               │   │       ├── SignatureProviderPort.java
│   │   │               │   │       ├── EventPublisherPort.java
│   │   │               │   │       └── SecretManagerPort.java
│   │   │               │   │
│   │   │               │   ├── exception/              # Domain Exceptions
│   │   │               │   │   ├── SignatureNotFoundException.java
│   │   │               │   │   ├── RuleValidationException.java
│   │   │               │   │   ├── ProviderException.java
│   │   │               │   │   ├── FallbackExhaustedException.java
│   │   │               │   │   └── DomainException.java
│   │   │               │   │
│   │   │               │   └── event/                   # Domain Events
│   │   │               │       ├── SignatureRequestCreated.java
│   │   │               │       ├── ChallengeSent.java
│   │   │               │       ├── ChallengeFailed.java
│   │   │               │       ├── ProviderFailed.java
│   │   │               │       ├── SignatureCompleted.java
│   │   │               │       ├── SignatureExpired.java
│   │   │               │       ├── SignatureAborted.java
│   │   │               │       └── DomainEvent.java
│   │   │               │
│   │   │               ├── application/                # APPLICATION LAYER
│   │   │               │   ├── usecase/                # Use Case Implementations
│   │   │               │   │   ├── StartSignatureUseCaseImpl.java
│   │   │               │   │   ├── ConfigureRuleUseCaseImpl.java
│   │   │               │   │   ├── RetryWithFallbackUseCaseImpl.java
│   │   │               │   │   ├── QuerySignatureUseCaseImpl.java
│   │   │               │   │   └── ManageConnectorUseCaseImpl.java
│   │   │               │   │
│   │   │               │   ├── dto/                    # DTOs
│   │   │               │   │   ├── request/
│   │   │               │   │   │   ├── CreateSignatureRequest.java
│   │   │               │   │   │   ├── CreateRuleRequest.java
│   │   │               │   │   │   └── UpdateConnectorRequest.java
│   │   │               │   │   └── response/
│   │   │               │   │       ├── SignatureResponse.java
│   │   │               │   │       ├── RuleResponse.java
│   │   │               │   │       ├── RoutingTimelineResponse.java
│   │   │               │   │       └── CostAnalysisResponse.java
│   │   │               │   │
│   │   │               │   ├── mapper/                 # DTO <-> Domain Mappers
│   │   │               │   │   ├── SignatureMapper.java
│   │   │               │   │   ├── RuleMapper.java
│   │   │               │   │   └── ConnectorMapper.java
│   │   │               │   │
│   │   │               │   └── validator/              # Business Validation
│   │   │               │       ├── SpelValidator.java
│   │   │               │       └── TransactionContextValidator.java
│   │   │               │
│   │   │               └── infrastructure/             # INFRASTRUCTURE LAYER
(Adapters)
│   │   │                   │
│   │   │                   ├── adapter/
│   │   │                   │   ├── inbound/           # Inbound Adapters (REST,
etc.)
│   │   │                   │   │   ├── rest/
│   │   │                   │   │   │   ├── SignatureController.java
│   │   │                   │   │   │   ├── AdminRuleController.java
│   │   │                   │   │   │   ├── AdminConnectorController.java
│   │   │                   │   │   │   └── HealthController.java
│   │   │                   │   │   ├── exception/
│   │   │                   │   │   │   ├── GlobalExceptionHandler.java
│   │   │                   │   │   │   └── ErrorResponse.java
│   │   │                   │   │   └── security/
│   │   │                   │   │       ├── SecurityConfig.java
│   │   │                   │   │       ├── IdempotencyFilter.java
│   │   │                   │   │       └── RoleBasedAccessControl.java
│   │   │                   │   │
│   │   │                   │   └── outbound/          # Outbound Adapters
│   │   │                   │       ├── persistence/   # JPA Repositories
│   │   │                   │       │   ├── jpa/
│   │   │                   │       │   │   ├── SignatureRequestJpaRepository.java
│   │   │                   │       │   │   ├── RoutingRuleJpaRepository.java
│   │   │                   │       │   │   ├── ConnectorConfigJpaRepository.java
│   │   │                   │       │   │   └── AuditLogJpaRepository.java
│   │   │                   │       │   ├── entity/
│   │   │                   │       │   │   ├── SignatureRequestEntity.java
│   │   │                   │       │   │   ├── SignatureChallengeEntity.java
│   │   │                   │       │   │   ├── RoutingRuleEntity.java
│   │   │                   │       │   │   ├── ConnectorConfigEntity.java
│   │   │                   │       │   │   ├── OutboxEventEntity.java
│   │   │                   │       │   │   └── AuditLogEntity.java
│   │   │                   │       │   ├── mapper/
│   │   │                   │       │   │   ├── SignatureEntityMapper.java
│   │   │                   │       │   │   ├── RuleEntityMapper.java
│   │   │                   │       │   │   └── ConnectorEntityMapper.java
│   │   │                   │       │   └── adapter/
│   │   │                   │       │       ├──
SignatureRequestRepositoryAdapter.java
│   │   │                   │       │       ├── RoutingRuleRepositoryAdapter.java
│   │   │                   │       │       ├──
ConnectorConfigRepositoryAdapter.java
│   │   │                   │       │       └── AuditLogRepositoryAdapter.java
│   │   │                   │       │
│   │   │                   │       ├── provider/      # Provider Adapters
│   │   │                   │       │   ├── SignatureProviderAdapter.java
│   │   │                   │       │   ├── twilio/
│   │   │                   │       │   │   ├── TwilioSmsProvider.java
│   │   │                   │       │   │   ├── TwilioConfig.java
│   │   │                   │       │   │   └── TwilioResponseMapper.java
│   │   │                   │       │   ├── push/
│   │   │                   │       │   │   ├── PushNotificationProvider.java
│   │   │                   │       │   │   └── PushConfig.java
│   │   │                   │       │   ├── voice/
│   │   │                   │       │   │   ├── VoiceCallProvider.java
│   │   │                   │       │   │   └── VoiceConfig.java
│   │   │                   │       │   └── biometric/
│   │   │                   │       │       ├── BiometricProvider.java (stub)
│   │   │                   │       │       └── BiometricConfig.java
│   │   │                   │       │
│   │   │                   │       ├── event/         # Event Publishing
│   │   │                   │       │   ├── OutboxEventPublisher.java
│   │   │                   │       │   ├── KafkaEventPublisher.java
│   │   │                   │       │   └── EventSerializer.java
│   │   │                   │       │
│   │   │                   │       └── secret/        # Secret Management
│   │   │                   │           ├── VaultSecretManager.java
│   │   │                   │           └── VaultConfig.java
│   │   │                   │
│   │   │                   ├── config/                # Spring Configuration
│   │   │                   │   ├── DomainConfig.java
│   │   │                   │   ├── ApplicationConfig.java
│   │   │                   │   ├── PersistenceConfig.java
│   │   │                   │   ├── KafkaConfig.java
│   │   │                   │   ├── ResilienceConfig.java
│   │   │                   │   ├── ObservabilityConfig.java
│   │   │                   │   └── OpenApiConfig.java
│   │   │                   │
│   │   │                   ├── resilience/            # Resilience4j Integration
│   │   │                   │   ├── CircuitBreakerManager.java
│   │   │                   │   ├── ProviderHealthMonitor.java
│   │   │                   │   └── DegradedModeManager.java
│   │   │                   │
│   │   │                   └── observability/         # Observability
│   │   │                       ├── logging/
│   │   │                       │   ├── MdcFilter.java
│   │   │                       │   └── StructuredLogger.java
│   │   │                       ├── metrics/
│   │   │                       │   ├── SignatureMetrics.java
│   │   │                       │   └── ProviderMetrics.java
│   │   │                       └── tracing/
│   │   │                           └── TracingConfig.java
│   │   │
│   │   └── resources/
│   │       ├── application.yml
│   │       ├── application-dev.yml
│   │       ├── application-prod.yml
│   │       ├── db/
│   │       │   └── migration/
│   │       │       ├── V1__initial_schema.sql
│   │       │       ├── V2__add_audit_log.sql
│   │       │       └── V3__add_indexes.sql
│   │       ├── kafka/
│   │       │   └── schemas/
│   │       │       ├── signature-event.avsc
│   │       │       └── audit-event.avsc
│   │       ├── openapi/
│   │       │   └── signature-router-api.yaml
│   │       └── logback-spring.xml
│   │
│   └── test/
│       ├── java/
│       │   └── com/
│       │       └── bank/
│       │           └── signature/
│       │               ├── domain/              # Pure Unit Tests
│       │               │   ├── model/
│       │               │   │   └── SignatureRequestTest.java
│       │               │   └── service/
│       │               │       ├── RoutingServiceTest.java
│       │               │       └── ChallengeServiceTest.java
│       │               │
│       │               ├── application/         # Use Case Tests
│       │               │   └── usecase/
│       │               │       ├── StartSignatureUseCaseTest.java
│       │               │       └── ConfigureRuleUseCaseTest.java
│       │               │
│       │               └── infrastructure/      # Integration Tests
│       │                   ├── adapter/
│       │                   │   ├── rest/
│       │                   │   │   └── SignatureControllerIT.java
│       │                   │   └── persistence/
│       │                   │       └── SignatureRepositoryIT.java
│       │                   │
│       │                   └── e2e/             # End-to-End Tests
│       │                       └── SignatureFlowE2ETest.java
│       │
│       └── resources/
│           ├── application-test.yml
│           ├── testcontainers/
│           │   └── docker-compose-test.yml
│           └── fixtures/
│               ├── sample-transaction-context.json
│               └── sample-routing-rules.json
│
├── admin-portal/                              # React Admin Portal
│   ├── package.json
│   ├── src/
│   │   ├── components/
│   │   │   ├── rules/
│   │   │   │   ├── RuleList.tsx
│   │   │   │   ├── RuleEditor.tsx
│   │   │   │   └── SpelValidator.tsx
│   │   │   ├── timeline/
│   │   │   │   └── RoutingTimeline.tsx
│   │   │   ├── dashboard/
│   │   │   │   ├── CostOptimization.tsx
│   │   │   │   └── MetricsDashboard.tsx
│   │   │   └── audit/
│   │   │       └── AuditLogViewer.tsx
│   │   ├── api/
│   │   │   └── signatureRouterClient.ts
│   │   └── App.tsx
│   └── README.md
```

```
│
├── docker/
│   ├── Dockerfile
│   ├── docker-compose.yml
│   └── postgres/
│       └── init.sql
│
└── k8s/
    ├── deployment.yaml
    ├── service.yaml
    ├── configmap.yaml
    └── secret.yaml
```

---

2. Layer Responsibilities

2.1 Domain Layer (Pure Business Logic)

Regla de Oro: Esta capa NO DEPENDE de nada. Cero imports de Spring, JPA, Kafka, etc.

Contiene:

- Aggregates, Entities, Value Objects
- Domain Services (lógica que no pertenece a un agregado)
- Port interfaces (contratos de lo que necesita el dominio)
- Domain Events (eventos de negocio)
- Domain Exceptions

No contiene:

- ❌ Annotations de JPA (`@Entity`, `@Table`)
- ❌ Annotations de Spring (`@Service`, `@Component`)
- ❌ Dependencias de HTTP, JSON, Base de datos
- ❌ Lógica de infraestructura

Ejemplo:

```java
// ✅ CORRECTO - Domain puro
public class SignatureRequest {
    private final UUID id;
    private final CustomerId customerId;
    private final TransactionContext transactionContext;
    private SignatureStatus status;
    private List<SignatureChallenge> challenges;
    // Business logic puro
    public void addChallenge(SignatureChallenge challenge) {
        if (this.status == SignatureStatus.EXPIRED) {
            throw new DomainException("Cannot add challenge to expired request");
        }
        this.challenges.add(challenge);
    }
}
```

2.2 Application Layer (Orchestration)

Responsabilidades:

- Implementar Use Cases (orquestar dominio)
- Coordinar transacciones
- Transformar DTOs ↔ Domain Models
- Validaciones de aplicación (no de negocio)

Usa:

- Ports del dominio
- Domain Services
- Aggregates

No contiene:

- ❌ Detalles de HTTP (StatusCode, Headers)
- ❌ Queries SQL directas
- ❌ Lógica de negocio (debe estar en dominio)

Ejemplo:

```
// ✅ CORRECTO - Use Case implementation
@Service
@Transactional
public class StartSignatureUseCaseImpl implements StartSignatureUseCase {

    private final SignatureRequestRepository repository;
    private final RoutingService routingService;
    private final SignatureProviderPort providerPort;
    private final EventPublisherPort eventPublisher;

    @Override
    public SignatureResponse start(CreateSignatureRequest dto) {
        // 1. Construir domain model
        TransactionContext context =
TransactionContext.from(dto.getTransactionDetails());
        SignatureRequest request = SignatureRequest.create(dto.getCustomerId(),
context);

        // 2. Evaluar routing (domain service)
        List<RoutingRule> rules = ruleRepository.findAllEnabled();
        ChannelType channel = routingService.evaluateRoute(context, rules);

        // 3. Crear challenge (domain logic)
        SignatureChallenge challenge = request.createChallenge(channel);

        // 4. Persistir aggregate
        repository.save(request);

        // 5. Llamar provider (outbound port)
        ProviderResult result = providerPort.sendChallenge(challenge);

        // 6. Actualizar estado
        challenge.markSent(result);

        // 7. Publicar evento
        eventPublisher.publish(new ChallengeSent(request.getId(),
challenge.getId()));

        // 8. Retornar DTO
        return SignatureMapper.toResponse(request);
    }
}
```

2.3 Infrastructure Layer (Technical Concerns)

Responsabilidades:

- Implementar adapters para ports del dominio
- Configuración de frameworks (Spring, JPA, Kafka)
- Concerns técnicos (logging, metrics, tracing)
- API REST Controllers
- Persistencia (JPA entities, mappers)
- Integración con sistemas externos

Ejemplo - Inbound Adapter (REST Controller):

```
@RestController
@RequestMapping("/api/v1/signatures")
```

```
public class SignatureController {

    private final StartSignatureUseCase startUseCase;

    @PostMapping
    public ResponseEntity<SignatureResponse> createSignature(
        @RequestHeader("Idempotency-Key") String idempotencyKey,
        @Valid @RequestBody CreateSignatureRequest request
    ) {
        SignatureResponse response = startUseCase.start(request);
        return ResponseEntity.status(HttpStatus.CREATED).body(response);
    }
}
```

Ejemplo - Outbound Adapter (JPA Repository):

```
@Component
public class SignatureRequestRepositoryAdapter implements
SignatureRequestRepository {

    private final SignatureRequestJpaRepository jpaRepository;
    private final SignatureEntityMapper mapper;

    @Override
    public void save(SignatureRequest domainModel) {
        SignatureRequestEntity entity = mapper.toEntity(domainModel);
        jpaRepository.save(entity);
    }

    @Override
    public Optional<SignatureRequest> findById(UUID id) {
        return jpaRepository.findById(id)
            .map(mapper::toDomain);
    }
}
```

---

3. Dependency Flow

```
┌─────────────────────────────────────────────────────┐
│                   REST Controllers                  │
│               (Infrastructure Layer)                │
└─────────────────────────────────────────────────────┘
                        │
                        │ calls
                        ▼
┌─────────────────────────────────────────────────────┐
│                    Use Case Impl                    │
│                (Application Layer)                   │
│  • Orchestrates domain                               │
│  • Uses domain ports                                 │
└────────┬──────────────────────────────────┬──────────┘
         │ uses                             │ uses
         ▼                                  ▼
┌─────────────────────┐          ┌─────────────────────┐
│  Domain Services    │          │     Aggregates      │
│  (Domain Layer)     │          │   (Domain Layer)    │
└─────────────────────┘          └─────────────────────┘
         │ defines                         │ defines
         ▼                                 ▼
┌──────────────────────────────────────────────────────┐
│                  Port Interfaces                     │
│                 (Domain Layer)                       │
│  • SignatureRequestRepository                        │
│  • SignatureProviderPort                             │
│  • EventPublisherPort                                │
└─────────────────────────┬────────────────────────────┘
                          │ implemented by
                          ▼
┌──────────────────────────────────────────────────────┐
│                 Adapters                             │
│            (Infrastructure Layer)                    │
│  • JPA Repositories                                  │
│  • Provider Clients (Twilio, etc.)                   │
│  • Kafka Publishers                                  │
└──────────────────────────────────────────────────────┘
```

Regla clave: Las dependencias apuntan HACIA el dominio, nunca desde el dominio hacia afuera.

---

4. Key Design Patterns

4.1 Repository Pattern

```java
// Port (Domain)
public interface SignatureRequestRepository {
    void save(SignatureRequest request);
    Optional<SignatureRequest> findById(UUID id);
    List<SignatureRequest> findByCustomerId(CustomerId customerId);
}

// Adapter (Infrastructure)
@Component
```

```
public class SignatureRequestRepositoryAdapter implements
SignatureRequestRepository {
    private final SignatureRequestJpaRepository jpaRepo;
    // Implementation using JPA
}
```

4.2 Port/Adapter Pattern

```
// Port (Domain - Outbound)
public interface SignatureProviderPort {
    ProviderResult sendChallenge(SignatureChallenge challenge);
    boolean verifyResponse(String challengeId, String userResponse);
}

// Adapter (Infrastructure)
@Component
public class SignatureProviderAdapter implements SignatureProviderPort {
    private final Map<ProviderType, SignatureProvider> providers;

    @Override
    public ProviderResult sendChallenge(SignatureChallenge challenge) {
        SignatureProvider provider = providers.get(challenge.getProvider());
        return provider.send(challenge);
    }
}
```

4.3 Strategy Pattern (Provider Selection)

```
// Common interface
public interface SignatureProvider {
    ProviderResult send(SignatureChallenge challenge);
    ProviderType getType();
}

// Implementations
@Component
public class TwilioSmsProvider implements SignatureProvider {
    // Twilio-specific logic
}

@Component
public class PushNotificationProvider implements SignatureProvider {
    // Push-specific logic
}
```

4.4 Outbox Pattern

```
@Service
@Transactional
public class OutboxEventPublisher implements EventPublisherPort {

    private final OutboxEventRepository outboxRepo;

    @Override
    public void publish(DomainEvent event) {
        // Save event to outbox table in SAME transaction
        OutboxEvent outboxEvent = OutboxEvent.from(event);
        outboxRepo.save(outboxEvent);

        // Debezium will pick it up and publish to Kafka
    }
}
```

---

5. Testing Strategy by Layer

5.1 Domain Layer Tests (Pure Unit Tests)

```
class SignatureRequestTest {

    @Test
    void shouldTransitionToSignedWhenChallengeCompleted() {
        // Given
        SignatureRequest request = SignatureRequest.create(customerId, context);
        SignatureChallenge challenge = request.createChallenge(ChannelType.SMS);

        // When
        challenge.markCompleted("provider-proof-123");
        request.completeSignature(challenge);

        // Then
        assertThat(request.getStatus()).isEqualTo(SignatureStatus.SIGNED);
    }
}
```

Características:

- ✅ No Spring Context
- ✅ No base de datos
- ✅ Súper rápidos (<1ms)
- ✅ Test lógica de negocio pura

5.2 Application Layer Tests (Use Case Tests)

```
@ExtendWith(MockitoExtension.class)
class StartSignatureUseCaseTest {

    @Mock private SignatureRequestRepository repository;
    @Mock private RoutingService routingService;
    @Mock private SignatureProviderPort providerPort;

    @InjectMocks
    private StartSignatureUseCaseImpl useCase;

    @Test
    void shouldCreateSignatureAndSendChallenge() {
        // Given
        when(routingService.evaluateRoute(any(), any()))
            .thenReturn(ChannelType.SMS);
        when(providerPort.sendChallenge(any()))
            .thenReturn(ProviderResult.success("provider-id-123"));

        // When
        SignatureResponse response = useCase.start(createRequest);

        // Then
        verify(repository).save(any(SignatureRequest.class));
        verify(providerPort).sendChallenge(any());
        assertThat(response.getStatus()).isEqualTo("CHALLENGE_SENT");
    }
}
```

Características:

- ✅ Mockear ports (dependencies)
- ✅ Test orquestación
- ✅ No infraestructura real

5.3 Infrastructure Layer Tests (Integration Tests)

```
@SpringBootTest
@Testcontainers
class SignatureRepositoryIT {

    @Container
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>
("postgres:15");
```

```
    @Autowired
    private SignatureRequestRepository repository;

    @Test
    void shouldPersistAndRetrieveSignatureRequest() {
        // Given
        SignatureRequest request = SignatureRequest.create(customerId, context);

        // When
        repository.save(request);
        Optional<SignatureRequest> retrieved =
repository.findById(request.getId());

        // Then
        assertThat(retrieved).isPresent();
        assertThat(retrieved.get().getCustomerId()).isEqualTo(customerId);
    }
}
```

Características:

- ✅ Testcontainers (PostgreSQL, Kafka)
- ✅ Test integración con infraestructura real
- ✅ Más lentos pero más realistas

5.4 End-to-End Tests

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@Testcontainers
class SignatureFlowE2ETest {

    @LocalServerPort
    private int port;

    @Test
    void shouldCompleteFullSignatureFlow() {
        // Given
        String idempotencyKey = UUID.randomUUID().toString();
        CreateSignatureRequest request = buildRequest();

        // When - Create signature
        ResponseEntity<SignatureResponse> response = restTemplate
            .exchange("/api/v1/signatures", POST,
                httpEntity(request, idempotencyKey),
                SignatureResponse.class);
        // Then
        assertThat(response.getStatusCode()).isEqualTo(CREATED);

        // And - Verify Kafka event was published
        ConsumerRecord<String, String> event =
kafkaConsumer.poll(Duration.ofSeconds(5));
        assertThat(event.value()).contains("CHALLENGE_SENT");
    }
}
```

---

6. Configuration by Environment

6.1 application.yml (Base)

```yaml
spring:
  application:
    name: signature-router
  profiles:
    active: ${SPRING_PROFILE:dev}

server:
  port: 8080
  shutdown: graceful

management:
  endpoints:
    web:
      exposure:
        include: health,info,metrics,prometheus
  metrics:
    export:
      prometheus:
        enabled: true
```

6.2 application-dev.yml

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/signature_dev
    username: dev_user
    password: dev_pass
  jpa:
    show-sql: true
    properties:
      hibernate:
        format_sql: true
```

```yaml
logging:
  level:
    com.bank.signature: DEBUG
```

6.3 application-prod.yml

```yaml
spring:
  datasource:
    url: ${DB_URL}
    username: ${DB_USER}
    password: ${DB_PASS}
    hikari:
      maximum-pool-size: 20
      minimum-idle: 5
      connection-timeout: 2000
  jpa:
    show-sql: false
    properties:
      hibernate:
        format_sql: false

logging:
  level:
    com.bank.signature: INFO
```

---

7. Key Dependencies (pom.xml excerpt)

```xml
<dependencies>
    <!-- Spring Boot 3 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>3.2.0</version>
    </dependency>

    <!-- Spring Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- PostgreSQL Driver -->
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
    </dependency>

    <!-- Kafka -->
    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka</artifactId>
    </dependency>

    <!-- Resilience4j -->
    <dependency>
        <groupId>io.github.resilience4j</groupId>
        <artifactId>resilience4j-spring-boot3</artifactId>
        <version>2.1.0</version>
    </dependency>

    <!-- Vault -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-vault-config</artifactId>
    </dependency>

    <!-- Observability -->
    <dependency>
        <groupId>io.micrometer</groupId>
        <artifactId>micrometer-registry-prometheus</artifactId>
    </dependency>
    <dependency>
        <groupId>io.micrometer</groupId>
        <artifactId>micrometer-tracing-bridge-brave</artifactId>
    </dependency>

    <!-- Testing -->
    <dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>postgresql</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>kafka</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Status: ✅ **COMPLETE - READY FOR IMPLEMENTATION**